****

An INTERACTIVE PROGRAMMING LANGUAGE for CONTROL of ROBOTS

by Lichen Wang

Dear Jim: *Received: 77-Oct-19*

Here are the "Robot Control Language" write up and program listings that I told you about. I also included a program written in this language by Gordon French and some comments by Gregory Yob. [NOTE: Greg's comments appear at the end of this documentation—TW] *Greg calls the language TINMAN; while I did not call it any name. If everything has to have an acronym, I would rather call it WSFN (Which Stands For Nothing).*

I also have a version of this language written in Control Basic (as opposed to Assembler Language as included).

Lichen Wang
150 Tennyson Ave
Palo Alto, CA 94301
(415) 321-6983

Introduction

This is a simple-minded "Tiny" language. The basic idea is to use an ASCII keyboard as the command input device for a microprocessor controlled robot. Rudimentary commands consisting of a single ASCII character are executed by the robot as soon as they are typed. Some primitive forms of IF-THEN-ELSE, REPEAT-n-TIMES, and run-time-macro are included. Paired parentheses are used to group any number of commands into a single command and enable one to construct complex structures. The run-time-macro is probably the most important feature that makes a simple language such as this one powerful and easy to use.

This language is open-ended and is not limited to any specific robot. To illustrate this point, a version of this language is implemented on an 8080-based microprocessor to control a robot who has a "mind" but no "body". Later in this article, two different "bodies" are attached to the robot's "mind" as examples. Revisions and implementations on other microprocessors can be made if there are such interests.

The Mind

In this implementation, one uses the VDM-1 interface and a video display to represent the robot's mind – so that one can always read what is in the robot's mind. Other hardware required are: an ASCII keyboard with interface, and an 8080 based microprocessor system with 8K of RAM at 0000-1FFF. The major part of the memory is used for the stack. The assembler listing and object code of the program are included following this article.

The robot's mind consists of an input line buffer of 58 characters long, 15 lines of macro definition space (also of 58 characters each) and a 4-decimal-digit accumulator. When one starts the program, the entire screen of the video display is cleared, the characters "A=0000" are displayed at the top right corner, and a cursor is shown at the top left corner (i.e., the beginning of the input line buffer). Whenever the cursor shows in the input line buffer, it means the robot is not doing anything but waiting for input from the keyboard. When the cursor disappears, the robot is executing a command. But even then, if one types on the keyboard, the robot will stop whatever it was doing and obey the new command.

There are three rudimentary commands for this bodyless robot. More will be added when a body exists.

␣	(space, blank) No operation
+	Increment the accumulator
–	Decrement the accumulator

The no-op command is sometimes needed in IF-THEN-ELSE commands discussed later. When one types the character "+" or "–", it replaces the cursor on the screen, the accumulator is changed accordingly, the "+" or "–" is erased, and then the cursor reappears. (But all these happen in a flash.) Attempts to increment beyond 9999 or decrement below 0000 will not change the content of the accumulator.

If one types the sequence of characters "(++–+)", the accumulator will be changed after the ")" character is typed, and not while the individual "+" and "–" are typed. The sequence stored and displayed on the input line buffer as entered, and the cursor is moved to the right at the same pace. If a RUB-OUT (or DEL) character is typed before the closing ")" is typed, the input buffer is erased and the partial command is not executed at all.

The bodyless robot has two IF-THEN-ELSE commands. One of them tests the accumulator. The other is a random decision maker:

T [then command] [else command]	Test if A>0
? [then command] [else command]	Randomly choose one

The microprocessor interface consists of one parallel input port and one parallel output port. One bit of the input port is connected to a flip-flop which in turn can be set by touching the micro-switch on the robot. Two other bits of the input port tell the micro-processor whether the stepping motors are ready to be pulsed. Five bits of the output port are used; one of them used to reset the sensor flip-flop, and two bits each – direction and step, for the stepping motors. Other bits of the input/output ports are for further expansion.

To control the body of this robot, one adds the following command to what was discussed in the previous section:

F	Move forward one step
B	Move backward one step
R	Rotate right one unit angle
L	Rotate left one unit angle
S	[then command] [else command] Test & reset the sensor

The "unit angle" is the smallest angle the robot can rotate. It is determined by the step size and the distance between the two driven wheels. The "S" command is an IF-THEN-ELSE command similar to the "T" and "?" commands discussed in the last section. The assembler listing and the object code of the modifications needed to implement these new commands are presented below.

Another Body

Another robot of a sort can be even easier to construct – if the micro-system has a DAZZLER. One uses software to

control an imaginary turtle on another video display connected to the DAZZLER. The black & white mode is used to get a 128 x 128 grid. The new commands added are:

F Move forward one grid cell
R Rotate 45 degrees right
H Go home (i.e., the center of screen)
N Face North (i.e., up)
W Leave white trace
B Leave black trace
C Clear screen

Note that the "B" command here is in conflict with the one in Section III. But since the robot has only one of these two alternative bodies, this does not cause any problem. The "C" clears the screen to all black. If "B" is in effect, "C" clears the screen to all white. When one marches the turtle off the boundary, it will re-enter the boundary on the opposite side of the screen. The assembler listing and the object code of the modifications needed to implement these new commands are presented below.

All IF-THEN-ELSE commands have this common syntax. The leading character, such as "T" or "?", specifies the IF condition, [then command] is the command to be executed by the robot when the IF condition is met, and [else command]. Thus: "??(case1) (case 2)? (case 3) (case 4)" has the same effect as: "?(?(case 1) (case 2)) (?(case 3) (case 4))" and will both cause the robot to randomly choose to execute one of the four cases.

The bodyless robot has two REPEAT-n-TIMES commands. One repeats a fixed number of times as specified, and the other uses the current value of the accumulator as the repeat count.

324 [command] repeats the [command] 324 times
A [command] repeats the [command] A number of times.

For example, "A—" will zero the accumulator, and "A+" will double the content of the accumulator. More complex sequence can be constructed using parentheses, e.g., 34 (6+12(+—+)A+).

Last, and by no means the least, is the macro definition command. It is used to define any character, which is not one of the basic commands, to be the equivalent of a simple or complex command.

D [ch] [command] Define [ch] to be [command]

For example, "A—" clears the accumulator, one can type "DZA—" and from there on simply type "Z" to clear the accumulator. One can also have "DX(Z34(6+12(+—+)A+))" for whatever purpose.

When one types in the definition, the sequence of characters are entered into the input buffer, and the cursor is moved to the right. When the last character (which completes the definition) is typed, the robot will search its mind to find if an old macro definition of that same character exists. In case an old definition exists, it will be marked to be deleted later. The robot then searches its mind again to find an empty line and copies the definition there. The examples above will be shown as "Z=A—" and "X=(Z34(6+12(+—+A+))", respectively.

If one attempts to define a character which already has a meaning in this language, such as "⌀", "+", "—", "T", "?", "A", "D", or any digit, the robot will erase the input buffer and will not do anything else. The RUB-OUT (or DEL) key can also be used to abort the definition without damaging the old definition if any. To erase an old macro definition without creating a new one, one types "D[ch]⌀". This is consistent with the convention that any undefined character is also a no-op.

The Body

One can construct a very simple robot similar to the well known "Turtle". Its moving parts consist of two stepping motors driving two independent wheels plus an idler wheel to keep its balance. A micro-switch attached to a bumper serving as its sensor and other "bells and whistles" can be added on later. The two driven wheels are arranged like the wheels on a barrow cart, and not like a bicycle. When both stepping motors are driven forward or backward, so is the robot. But when the motors are driven in opposite directions, the robot rotates.

Programming Examples

The following are a few examples of programs written in this language. These examples are heavily biased toward graphics and recursive macro definitions. This should not be interpreted as representing the strength or limitation of this language. One reason for using graphic examples is that the action of a mechanical robot such as the one described in Section III is more difficult to represent here than the graphics. Another reason is that this language has not been tested on a mechanical robot yet.

There are very few programming languages that allow recursive definitions, and many hobbyists think recursive routines are difficult to understand. There are many indications that the language in which one learns to express one's ideas profoundly influences one's process of thinking. It can be shown that recursion is really not any more difficult to understand than iteration.

Example 1: A walking measuring tape

Using the robot described above, one can define the macros:

DM (Sb(FM)B+)
DZ (A—Sbb9999R)

Now if one types the character "Z", the robot will clear its accumulator, reset its sensor, and start to rotate. When the robot is facing the direction that one wants to measure, one types the character "M" and the robot will stop rotating and start to execute the macro "M". This process will repeat until the robot finally touches something, it will go back one step, increment the accumulator, and return from the current level of macro "M". Since the macro "M" has been nested, the robot will step back and increment accumulator as many times as it had stepped forward. This means the robot will find its way back to where it started from, and the accumulator will show the distance to the obstacle in units of the robot's step size.

Example 2: Windmill

Using the robot described earlier, one can define macros to draw all kinds of windmills:

DX (CA-8(HNARP+))
DP (. . . . .)

Where "P" is a macro defined by the user to draw some curve. "P" should use only the "F", "R", and fixed number repeat commands and paired parentheses. After one defines "P", one can type "X" to draw the windmill. The robot will first clear the screen, zero the accumulator, and then repeat 8 times drawing the pattern "P", each time starting at the home position but facing a different direction. The accumulator will be incremented each time after "P" is drawn, and will be used to determine the direction of the next iteration.

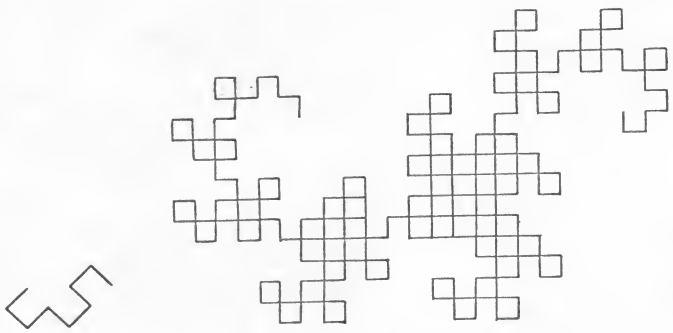
Example 3: Dragon curve [1]

The dragon curve is a family of very interesting curves. Figure 1A shows the 3rd order dragon curve and Fib. 1B shows the 8th order one. These curves can be constructed by

folding a strip of paper. One keeps doubling the strip and folds it into half the current length for n times – n being the order of the curve. One then opens all creases (there are 2**n–1 of them) to 90 degrees each, and a dragon curve is formed. The robot of Section IV can also draw these curves – up to the 12th order due to the 128 x 128 resolution. One defines:

```
DLT (–L6RJ+) G
DJT (–L2RJ+) G
DG4F
DQ (HN6R30FCF4RARL)
```

After these are defined, one sets the accumulator to the desired order (0 through 8) and types the character "Q". Recall that one can zero the accumulator by typing "A–", and that one can then set the accumulator to n by typing "n+". The macro "G" defines the segment size of the dragon. One can redefine "G" to be a smaller number of "F"s so that higher order dragons can fit the screen. With "DGF", up to the 12th order dragon can be drawn.



A B

Figure 1. Dragon curve of (A) order 3, and (B) order 8.

The dragon curve is defined by the macro "L", while macro "J" defines the reversed dragon curve. When folding the paper strip, one may observe the fact that the nth order dragon consists of a (n–1)th order dragon, a 90 degree fold, and another (n–1)th order dragon in reverse. The definition of "L" and "J" then becomes obvious when one adds the fact that the 0th order dragon is a plain strip of paper which is defined as "G". The macro "Q" is used to pick a starting position, clear the screen and pick a starting direction.

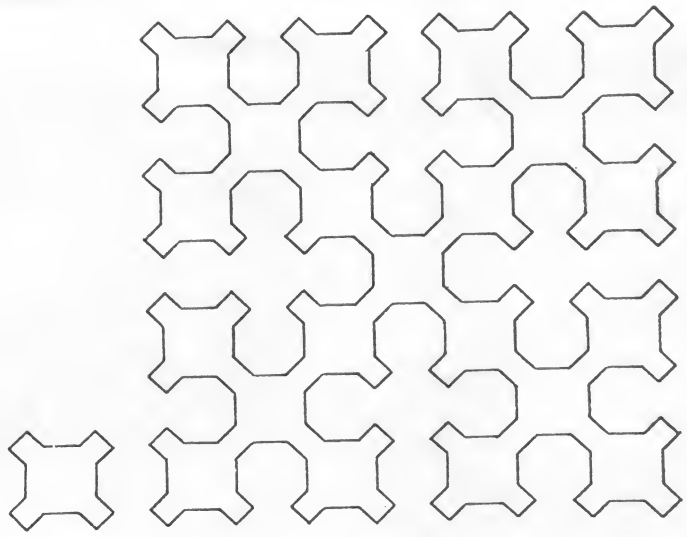
Example 4: Sierpinski curve [2]

Sierpinski curve of order 1 and 3 are shown in Fig. 2A and 2B respectively. Like Example 3, one may define:

```
DIT (–I2FI5RG5RI2FI+) 2R
DG4F
DY (HN63F2R61FRC4 (2FI))
```

After these are defined, one sets the accumulator to the desired order and types "Y" to draw the curve. Again, "G" can be redefined to a smaller number of "F"s to draw higher order "Y"s. For "DG0F", the accumulator can be set to 5.

The Sierpinski curve is closed and consists of four identical sides arranged in different directions and connected by a short line "2F". The sides are defined (recursively again) in the macro "I". and the closed curve itself is defined in the last part of the macro "Y", namely: "4(2FI)".



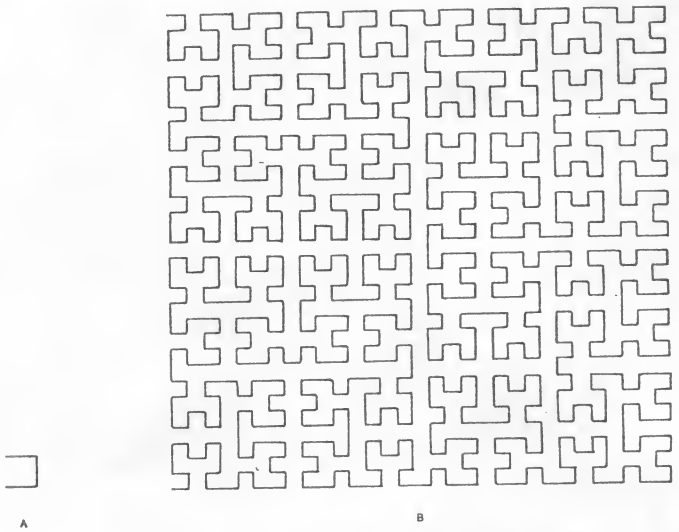
A B

Figure 2. Sierpinski curve of (A) order 1, and (B) order 3.

Example 5: Hilbert curve [2]

Hilbert curve of 1st and 5th order are shown in Fig. 3A and 3B respectively. The macro definitions are presented without further explanation:

```
DUT (–VG6RU2RGUG6RV+) 6R
DVT (–U2RGVG6RV2RGU+) 2R
DG4F
D$ (HN63F2R63FC2RU)
```



A B

Figure 3. Hilbert curve of (A) order 1, and (B) order 5.

Discussion

In Section II, only a very limited number of basic commands, a total of 8, are included and yet it has been demonstrated that quite complicated tasks can be accomplished by this simple language. In practice, one would put in more basic commands to make the language even easier to use. One would also want to include more facilities, e.g., variables or a stack for the accumulator, etc. As is evident when one glances through the program listings which follow, these additions can be easily made.

Footnotes:

1. See Scientific American April 1967 pages 116-123.
2. See Scientific American December 1976 pages 124-133.

One question that may arise is the limited number of characters in the ASCII code. If the keyboard is equipped with SHIFT and CNTRL keys, there are 128 different characters; and if the parity bit is not used and can be controlled from the keyboard, there are 256 different codes. One can further expand this by defining a few characters to be special escape characters. When the character table lookup routine encounters one of these escape characters, it will get the next character and scan a different table.

As the title of this article implies, the scope currently is limited to "interactive control"; there is no provision for artificial intelligence. Another limitation of this language in its current state is that there is no way to handle concurrent actions and real time interrupts from the robot. The author looks forward to feedbacks from the readers. He especially welcomes ideas which would enlarge the scope of the language as opposed to what specific command should be included or deleted.

The author wishes to thank Harvey A. Cohen and Marvin R. Winzenread for many interesting discussions and their pioneer works and publications in related topics that led to the development of this language.

□□□□

SOME SUPPLEMENTARY OBSERVATIONS FROM GREG YOB

To Lichen Wang
Re Tiny Robot Language (TINMAN) *Received:77-Oct-19*

Your design is hard to beat for conciseness. I found it a bit difficult to follow the examples, but that would be clearer with a live machine in front of me. All in all, a very clear and clean approach.

My first thoughts were mostly cosmetic in nature, ie, I prefer to be able to give multicharacter names, and locating the various tests embedded in a command string was sometimes confusing. Of course, tests can have parentheses appropriately located to mark them, and multicharacter command names or macro names could be done with a delimiter convention, ie

TURTLE

Tests could be either:

TEST (first) (second)

or

T! (first) (second)! note that (xxx) stands for whatever goes there..

As I said, these are cosmetic in nature . . .

In pondering the addition of bodies, I noted that the commands for the bodies outnumbered those for the mind. Since the macro facility is essential to the language, why not use it to implement each body rather than have each body eat up more characters in the ASCII set . . . The beginnings of such an approach would look like this (bear in mind I have not run it all the way through to be consistent with other aspects of TINMAN). Two new commands should take care of it:

I - Input from robot & place value in Accumulator
O (number or A) Output value to Robot

Of course, a stack gets pretty important, so some kind of push & pop commands may help out as well . . . Bear in mind I am not just adding some more commands, I am using these to remove all commands that ultimately are input and output commands.

With I, O, push & pop, then the other commands for a body may be written as macros. While we are at it, it might be neat to "store" screenfuls of macros in other sections of memory or on an external device, so that those used for a body would not clutter the VDM display.

A hint of an idea towards interactive & real-time uses would be to have the robot's input actually be a macro call. That is, the value returned by the robot would be a defined macro which is then executed. This would either be an interrupt (god forbid!!) or a polling kind of thing where after each operation, the inputs are polled, and if one needs treatment, the state of the language processor is stacked, and the required macro is called. A suitable form might look like:

P (robot #) (macro or command) or A	Poll robot and do command if ready, command can then input whatever value from robot, etc.

Of course, if robots appear, so must numbers for I and O.

Two methods may be used, one is to have the polling after input of a command from the keyboard, and the other is to have it at intervals within the program, say after each 20 executions of a mind primitive routine from the commands dispatcher, and after each 1000 passes through the keyboard input wait loop.

A naggy detail is that a robot may want some attention from the keyboard. This could be done, and would allow keyboard input during a run of a command by a slightly different version of I and O:

I -	Input number from keyboard & put in accumulator
I number or A	Input from Robot number FOO (The keyboard is Robot O)
O value or A	Write value on VDM in commands input buffer spot or, preferably, bottom line
O value or A (blank) value or A	Write first argument to Robot indicated by second argument.

This gets me into more tangles than I started with. However, I offer these as suggestions towards some of the problems you mentioned in your document.

I am of course interested in getting Source & Object for playing with. Meanwhile, thanks for an interesting afternoon of thought. . .

Gregory Yob
PO Box 354
Palo Alto, CA 94301
(415) 326-4039)

```
ASSEMBLER LISTING OF "THE MIND"

              ****************************************
              *
              *              W S F N
              *       (Which Stands For Nothing)
              *          ----------------
              *   An Interactive Programming Language
              *          for Control of Robots
              *       Version Ø.1, September 1977
              *            By Lichen Wang
              *
              ****************************************
              *
              * We have 8K of RAM at ØØØØ-1FFF, a VDM
              * at CCØØ-CFFF, and an ASCII keyboard.
              *
ØØØØ                 ORG   ØØØØØH
Ø29D          DANGER EQU   LAST+64     *Don't let stack
FD63          STKLMT EQU   -DANGER     *eat up program.
1FFE          STACK  EQU   Ø1FFEH
1FFE          SEED   EQU   STACK       *Seeds for RNDB
ØØC8          VDM    EQU   ØC8H        *Standard VDM-1
CCØØ          SCREEN EQU   ØCCØØH      *port & addr.
ØØØØ          KEYSTA EQU   ØØØH        *Status port,
ØØ8Ø          KEYDAV EQU   Ø8ØH        *key ready bit,
ØØØ1          KEYDAT EQU   ØØ1H        *& data port.
              *
```

```
                *   *** In the beginning ***
                *
0000 310020             LXI  SP,SEED+2   *Seeding RNDB
0003 CD5C02             CALL INIT        *Baptize the
0006 AF                 XRA  A           *robot, and
0007 D3C8               OUT  VDM         *brain wash it
0009 2100CC             LXI  H,SCREEN
000C 3ED0               MVI  A,(SCREEN+1024)/256
000E 3620           I1  MVI  M,' '
0010 23                 INX  H
0011 BC                 CMP  H
0012 C20E00             JNZ  I1
0015 21413D             LXI  H,'A='      *Put 'A=0000'
0018 223ACC             SHLD SCREEN+58   *in his mind
001B 213030             LXI  H,'00'      *(at the top-
001E 223CCC             SHLD SCREEN+60   *right corner).
0021 223ECC             SHLD SCREEN+62
                *
                *   *** DIRECT ***
                *
                * This is the main loop that clears
                * garbage (old definitions), reads a
                * command from the keyboard, puts it in
                * the robot's input buffer (which is the
                * top line on the VDM), and then let the
                * robot execute the command.
                *
0024 31FE1F     DIRECT LXI  SP,STACK
0027 2100CC     D1     LXI  H,SCREEN     *Clear the input
002A 063A       D2     MVI  B,58         *buffer,
002C 3620       D3     MVI  M,' '
002E 23                INX  H
002F 05                DCR  B
0030 C22C00            JNZ  D3
0033 0E7F              MVI  C,@DEL       *and other lines
0035 CDAE00            CALL FIND         *that has been
0038 CA2A00            JZ   D2           *marked.
003B 60                MOV  H,B          *B=0, just copy
003C 68                MOV  L,B          *From keyboard
003D 1100CC            LXI  D,SCREEN     *To input buffer
0040 CD5000            CALL RCMD
0043 2100CC            LXI  H,SCREEN     *From buffer
0046 50                MOV  D,B          *To nowhere
0047 58                MOV  E,B          *(DE=0)
0048 06FF              MVI  B,0FFH       *Now execute it
004A CD5000            CALL RCMD
004D C32700            JMP  D1           *Repeat all this
                *
                *   *** RCMD *** & *** CMD ***
                *
                * This subroutine is the syntax scanner.
                * If B reg. is 0, it scans for one
                * command and returns.  If B reg. is not
                * 0, it scans and executes one command.
                *
                * CMD is another entry, which assumes
                * that the first character of the
                * command is already in C reg.
                *
0050 CD8100     RCMD   CALL GETCH        *Get character
0053 DB00       CMD    IN   KEYSTA       *But did the
0055 E680              ANI  KEYDAV       *master have
0057 C22400            JNZ  DIRECT       *other wishes?
005A E5                PUSH H
005B CDC600            CALL RESRV        *Table lookup
005E 78                MOV  A,B          *for reserved
005F C26500            JNZ  C1           *Not reserved
0062 B7                ORA  A            *Reserved
0063 E3                XTHL             *Go do that
0064 C9                RET               *B is in Z flag
0065 B7         C1     ORA  A            *Not reserved
0066 E1                POP  H
0067 C8                RZ                *Don't execute
0068 E5                PUSH H            *Excecute user
0069 CDAE00            CALL FIND         *defined command
006C C27F00            JNZ  C2           *Undefined
006F E5                PUSH H            *Defined
0070 2163FD            LXI  H,STKLMT     *But let's see
0073 39                DAD  SP           *if there's more
0074 7C                MOV  A,H          *room in stack.
0075 B7                ORA  A
0076 FA2400            JM   DIRECT       *Forget it
0079 E1                POP  H            *OK, restore HL
007A 23                INX  H            *points to user
007B 23                INX  H            *definition
007C CD5000            CALL RCMD         *Scan that
007F E1         C2     POP  H            *Undefined ch.
0080 C9                RET               *is a No-op
                *
                *   *** GETCH ***
                *
                * This subroutine gets a character to C
                * reg., and optionally make a copy.  HL
                * points to where to get the character
                * from.  But if HL=0, we input from the
                * keyboard.  DE points to where to make
                * the copy.  But if DE=0, no copy is
                * made.
                *
0081 7B         GETCH  MOV  A,E          *Pointer to buff
0082 E63F              ANI  63
0084 FE39              CPI  57           *End of buffer?
0086 D22400            JNC  DIRECT       *Buffer overflow
0089 B2                ORA  D            *Make copy?
008A CA9000            JZ   G1           *No copy, fine
008D 3E7F              MVI  A,@DEL       *Copy, show
008F 12                STAX D            *cursor
0090 7C         G1     MOV  A,H          *Where from?
0091 B5                ORA  L
0092 C2A500            JNZ  G3           *From memory
0095 CD4601     G2     CALL RNDB         *(Randomlize)
0098 DB00              IN   KEYSTA       *From keyboard
009A E680              ANI  KEYDAV
009C CA9500            JZ   G2
009F DB01              IN   KEYDAT
00A1 4F                MOV  C,A
00A2 C3A700            JMP  G4
00A5 4E         G3     MOV  C,M          *From memory
```

```
ØØA6 [illegible]          INX  H
ØØA7 7A          G4       MOV  A,D           *Copy?
ØØA8 B3                   ORA  E
ØØA9 C8                   RZ                 *No copy, done
ØØAA 79                   MOV  A,C           *Copy it
ØØAB 12                   STAX D
ØØAC 13                   INX  D
ØØAD C9                   RET
                 *
                 *   *** FIND ***
                 *
                 * This subroutine searches the robot's
                 * memory to find a char.  It compares
                 * the C reg. with the first character
                 * of the 2nd through 16th line on the
                 * VDM screen.  If a match is found, the
                 * Z flag is set, and HL points to where
                 * it found it.  If no match, Z is reset.
                 *
ØØAE D5          FIND     PUSH D
ØØAF C5                   PUSH B
ØØBØ 214ØCC               LXI  H,SCREEN+64 *Ch.1 of line 2
ØØB3 114ØØØ               LXI  D,64        *Line length
ØØB6 Ø6ØF                 MVI  B,15        *Total lines
ØØB8 79                   MOV  A,C         *Character
ØØB9 BE          F1       CMP  M
ØØBA CAC3ØØ               JZ   F2          *Match found
ØØBD 19                   DAD  D           *No, next line
ØØBE Ø5                   DCR  B
ØØBF C2B9ØØ               JNZ  F1          *More lines
ØØC2 Ø4                   INR  B           *No, reset Z
ØØC3 C1          F2       POP  B
ØØC4 D1                   POP  D
ØØC5 C9                   RET
                 *
                 *   *** RESRV *** & *** DIGIT ***
                 *
                 * This subroutine checks the character
                 * in C reg. to see if it is one of the
                 * reserved characters.  If it is, the Z
                 * flag is set and the HL reg. is loaded
                 * with the address of the handler of
                 * that reserved character.  If C reg.
                 * does not contain one of the reserved
                 * characters, the Z flag is reset.
                 *
                 * The entry DIGIT will check if C reg.
                 * contains a digit.  It sets/resets Z
                 * flag the same way but does not change
                 * HL reg.
                 *
ØØC6 C5          RESRV    PUSH B
ØØC7 Ø6ØA                 MVI  B,TBLSZ     *# of entries
ØØC9 213EØ2               LXI  H,TABL      *& addr of table
ØØCC 79                   MOV  A,C         *Character
ØØCD BE          V1       CMP  M
ØØCE 23                   INX  H
ØØCF CAE6ØØ               JZ   V2          *Match found
ØØD2 23                   INX  H           *No
ØØD3 23                   INX  H           *Skip addr
ØØD4 Ø5                   DCR  B
ØØD5 C2CDØØ               JNZ  V1          *More in table
ØØD8 C1                   POP  B           *No, not there
ØØD9 2117Ø2               LXI  H,CMDØ9     *May be a digit
ØØDC 3E39        DIGIT    MVI  A,'9'       *Digit Ø-9?
ØØDE B9                   CMP  C
ØØDF D8                   RC               *No
ØØEØ 79                   MOV  A,C         *May be
ØØE1 D63Ø                 SUI  'Ø'
ØØE3 D8                   RC               *No
ØØE4 BF                   CMP  A           *Yes, set Z
ØØE5 C9                   RET
ØØE6 7E          V2       MOV  A,M         *Matched table
ØØE7 23                   INX  H
ØØE8 66                   MOV  H,M         *Load HL from
ØØE9 6F                   MOV  L,A         *table
ØØEA C1                   POP  B
ØØEB C9                   RET
                 *
                 ****************************************
                 *
                 * From here on are the routines that
                 * handle each reserved characters, and
                 * a table of these characters.
                 *
                 * RCMD or CMD will use the table and
                 * may jump to one of these routines.
                 * The Z flag is set if we are in the
                 * non-execute mode.  Reg. B also will
                 * be Ø in this case.  Reg. C has the
                 * character to be handled.  HL points
                 * to where the command string is from.
                 * If HL=Ø, command is from keyboard.
                 * DE points to where a copy of the
                 * command string is to be made.
                 * If DE=Ø, no copy is going to be made.
                 *
                 * Each handler should end with a RET and
                 * reg. BDEHL should not be destroyed.
                 *
                 ****************************************
                 *
                 *   *** blank ***
                 *
                 * A blank (space) is a no-op.
                 *
ØØEC C9          CMDBL  RET                *No-op
                 *
                 *   *** D ***
                 *
                 * D[ch][command]
                 *
                 * This command defines the [ch] to be
                 * the equivalence of the [command].
                 * [ch] must not be one of the reserved
                 * characters.
                 *
                 * If old definition exists, the old one
```

```
00ED CD8100
00F0 E5
00F1 CDC600
00F4 CA2400
00F7 78
00F8 B7
00F9 C20001
00FC E1
00FD C35000
0100 CDAE00
0103 C20801
0106 367F
0108 C5
0109 0E20
010B CDAE00
010E C22400
0111 C1
0112 71
0113 23
0114 363D
0116 23
0117 EB
0118 E3
0119 0600
011B CD8100
011E 3E20
0120 B9
0121 C22901
0124 1B
0125 1B
0126 12
0127 1B
0128 12
0129 CD5300
012C D1
012D 06FF
012F C9
```

```
* will be marked and will be deleted
* by DIRECT.  If the new definition
* is No-op ' ', it will not be stored
* in the robot's mind.  Any undefind
* character is a No-op anyway.
*
CMDD    CALL GETCH
        PUSH H
        CALL RESRV          *Is ch reserved?
        JZ   DIRECT         *Yes, forget it
        MOV  A,B            *No
        ORA  A              *Execute?
        JNZ  CD1            *Yes
        POP  H              *No, non-execute
        JMP  RCMD           *Skip it
CD1     CALL FIND           *Execute
        JNZ  CD2            *No old def.
        MVI  M,@DEL         *Mark old def.
CD2     PUSH B              *Now, new def.
        MVI  C,' '          *Find space in
        CALL FIND           *robot's mind
        JNZ  DIRECT         *No luck
        POP  B              *Found room
        MOV  M,C            *Tell him the ch
        INX  H
        MVI  M,'='          *is defind as
        INX  H
        XCHG
        XTHL
        MVI  B,0
        CALL GETCH          *Make copy
        MVI  A,' '          *But ' ' means
        CMP  C              *erase old one
        JNZ  CD3            *It's not ' '
        DCX  D              *It is ' '
        DCX  D              *erase the '='
        STAX D
        DCX  D              *and the ch
        STAX D
CD3     CALL CMD            *Continue scan
        POP  D              *for command
        MVI  B,0FFH         *Restore B
        RET
*
*   *** ( ***
*
* ([command]....)
*
* Any number of commands can be grouped
* together to be treated as one command
* by a pair of parentheses.  This can
* also be nested.
*
CMDLP   CALL GETCH
        MOV  A,C
        CPI  ')'
        RZ
        CALL CMD
        JMP  CMDLP
```

```
0130 CD8100
0133 79
0134 FE29
0136 C8
0137 CD5300
013A C33001
```

```
013D CA5C01
0140 CD4601
0143 C37001
0146 E5
0147 2AFE1F
014A 29
014B 7D
014C D25201
014F EE2D
0151 6F
0152 22FE1F
0155 E1
0156 E601
0158 C9
```

```
0159 C26201
015C CD5000
015F C35000
0162 E5
0163 2A3CCC
0166 7C
0167 B5
0168 2A3ECC
016B B4
016C B5
016D E1
016E FE30
0170 CA7E01
0173 CD5000
0176 0600
0178 CD5000
017B 06FF
017D C9
017E 0600
0180 CD5000
```

```
*
*   *** ? ***
*
* ?[then command][else command]
*
* This command tells the robot to
* randomly pick between the [then
* command] and the [else command].
*
* RNDB is a random bit generator.
*
CMDQM   JZ   SKIP           *Non-execute
        CALL RNDB           *Execute
        JMP  TEST
RNDB    PUSH H              *Random bit, see
        LHLD SEED           *Tausworthe,
        DAD  H              *Math. Comp. 19
        MOV  A,L            *(1965) 201-209
        JNC  CQ1
        XRI  02DH           *Watson et.al.,
        MOV  L,A            *Math. Comp. 16
CQ1     SHLD SEED           *(1962) 368-369
        POP  H
        ANI  1
        RET
*
*   *** T ***
*
* T[then command][else command]
*
* This command tests the robot's acc.
* If it is not 0, then execute the
* [then command], else execute the
* [else command]
*
* The SKIP and the TEST entries can
* be used by other commands of the
* same syntax.
*
CMDT    JNZ  CT1            *Execute
SKIP    CALL RCMD           *Non-execute
        JMP  RCMD
CT1     PUSH H              *Execute
        LHLD SCREEN+60      *Get acc.
        MOV  A,H            *Or all 4 digits
        ORA  L
        LHLD SCREEN+62
        ORA  H
        ORA  L
        POP  H
        CPI  '0'            *All '0'?
TEST    JZ   T1             *Test flag
        CALL RCMD           *Do "Then"
        MVI  B,0
        CALL RCMD           *Skip "Else"
        MVI  B,0FFH
        RET                 *Finished
T1      MVI  B,0            *Otherwise ...
        CALL RCMD           *Skip "Then"
```

```
0183 06FF             MVI  B,0FFH
0185 C35000           JMP  RCMD        *Do "Else"
                *
                *   *** + *** and *** - ***
                *
                * These are single character commands
                * which adds/subtracts 1 to/from the
                * accumulator respectively.
                *
0188 C8       CMDPS   RZ               *Non-execute
0189 E5               PUSH H           *Execute
018A CDE501           CALL GETA        *Get accumulator
018D E1               POP  H           *Value of acc
018E 3E27             MVI  A,9999/256  *Is it 9999?
0190 BC               CMP  H
0191 C29C01           JNZ  CPS1        *No
0194 3E0F             MVI  A,9999 MOD 256
0196 BD               CMP  L
0197 C29C01           JNZ  CPS1        *No
019A E1               POP  H           *Yes, don't add
019B C9               RET
019C 23       CPS1    INX  H           *Not 9999, add
019D 0E08     PUTA    MVI  C,8         *Put acc back
019F 29       P1      DAD  H           *Binary->decimal
01A0 7C               MOV  A,H
01A1 D664             SUI  100         *Divide by 100
01A3 DAA801           JC   P2
01A6 67               MOV  H,A
01A7 23               INX  H
01A8 0D       P2      DCR  C
01A9 C29F01           JNZ  P1          *Divide loop
01AC 4C               MOV  C,H         *Remainder
01AD 7D               MOV  A,L         *Quotient
01AE 213CCC           LXI  H,SCREEN+60 *Hi-order digit
01B1 CDBB01           CALL TWOD        *Put 2 digits
01B4 23               INX  H
01B5 79               MOV  A,C         *Low order
01B6 CDBB01           CALL TWOD        *2 more digits
01B9 E1               POP  H
01BA C9               RET
01BB 362F     TWOD    MVI  M,'0'-1     *One before '0'
01BD 34       P3      INR  M           *Make it '0'
01BE D60A             SUI  10
01C0 D2BD01           JNC  P3          *Count in 10's
01C3 C63A             ADI  '0'+10
01C5 23               INX  H
01C6 77               MOV  M,A         *and 1's
01C7 C9               RET
01C8 C8       CMDMS   RZ               *Non-execute
01C9 E5               PUSH H           *Execute
01CA CDE501           CALL GETA        *Get acc
01CD E1               POP  H
01CE 7C               MOV  A,H         *Is it 0?
01CF B5               ORA  L
01D0 2B               DCX  H
01D1 C29D01           JNZ  PUTA        *No, do -1
01D4 E1               POP  H           *Yes, don't -1
01D5 C9               RET
                *
```

```
                *   *** A ***
                *
                * A[repeated command]
                *
                * This command repeats the [repeated
                * command] A times.  A is the current
                * value of the robot's accumulator.
                *
01D6 CA5000   CMDA    JZ   RCMD        *Non-execute
01D9 E5               PUSH H           *Execute
01DA CDE501           CALL GETA        *Get acc
01DD E1               POP  H           *Value of acc
01DE E3               XTHL             *Old HL restored
01DF CD8100           CALL GETCH       *Next character
01E2 C32302           JMP  ITER        *Go repeat it
                *
                *   *** GETA *** & *** NUMB ***
                *
                * Subroutines to get the value of acc/
                * numbers from input.
                * HL points to where numbers are from.
                * Value is returned on top of the
                * stack. (Yes, top of the stack.)
                *
                * NUMB assumes the first digit is
                * already in C reg.
                *
01E5 213CCC   GETA    LXI  H,SCREEN+60 *Hi-order digit
01E8 CD8100           CALL GETCH       *of accumulator
01EB E5       NUMB    PUSH H           *Save old HL
01EC 210000           LXI  H,0         *HL=value now
01EF D5       N1      PUSH D           *Decimal->binary
01F0 3E18             MVI  A,24
01F2 BC               CMP  H           *H too big?
01F3 D2F701           JNC  N2          *No, OK
01F6 67               MOV  H,A         *Yes, set it
01F7 54       N2      MOV  D,H         *Save HL in DE
01F8 5D               MOV  E,L
01F9 29               DAD  H           *to do *10
01FA 29               DAD  H
01FB 19               DAD  D
01FC 29               DAD  H           *Got it
01FD 1600             MVI  D,0
01FF 79               MOV  A,C         *C = next digit
0200 D630             SUI  '0'
0202 5F               MOV  E,A
0203 19               DAD  D           *Add next digit
0204 D1               POP  D
0205 E3               XTHL             *Old HL back
0206 CD8100           CALL GETCH
0209 CDDC00           CALL DIGIT       *More digits?
020C E3               XTHL             *HL=value again
020D CAEF01           JZ   N1          *Yes, continue
0210 33               INX  SP          *No, dig out
0211 33               INX  SP          *return addr.
0212 E3               XTHL             *Put value there
0213 3B               DCX  SP          *Bury it
0214 3B               DCX  SP
0215 E3               XTHL             *Get old HL back
```

```
0216 C9                 RET
                *
                *   *** number ***
                *
                * [number][repeated command]
                *
                * This command will repeat the [repeated
                * command] any number of times as
                * specified.
                *
                * The entry ITER may be called by other
                * handlers of the same syntax.
                *
0217 CDEB01     CMD09   CALL NUMB       *Get value
021A 78                 MOV  A,B        *Execute?
021B B7                 ORA  A
021C C22302             JNZ  ITER       *Yes, iterate
021F F1                 POP  PSW        *No, pop value
0220 C35300             JMP  CMD        *Skip command
0223 E3         ITER    XTHL            *Iterate
0224 7C                 MOV  A,H        *HL has value
0225 B5                 ORA  L
0226 2B                 DCX  H
0227 C23302             JNZ  IT1        *Not 0, go on
022A E1                 POP  H          *It's 0
022B 0600               MVI  B,0        *Skip command
022D CD5300             CALL CMD
0230 06FF               MVI  B,0FFH     *Restore B
0232 C9                 RET
0233 E3         IT1     XTHL            *Iterations
0234 E5                 PUSH H
0235 C5                 PUSH B
0236 CD5300             CALL CMD
0239 C1                 POP  B
023A E1                 POP  H
023B C32302             JMP  ITER
                *
                *   *** TABL ***
                *
                * This is a table of all reserved
                * chracters (except digits, which are
                * also reserved).  Each reserved char.
                * is followed by the addr. of the
                * routine that handles the character.
                *
                * TBLSZ is the number of entries in
                * the table.
                *
023E 20         TABL    DB   ' '          *No-operation
023F EC00               DW   CMDBL
0241 44                 DB   'D'          *Define
0242 ED00               DW   CMDD
0244 28                 DB   '('          *Begin
0245 3001               DW   CMDLP
0247 29                 DB   ')'          *End
0248 2400               DW   DIRECT
024A 7F                 DB   @DEL         *Abort
024B 2400               DW   DIRECT
024D 3F                 DB   '?'          *Random choice
024E 3D01               DW   CMDQM
0250 54                 DB   'T'          *Test acc.
0251 5901               DW   CMDT
0253 2B                 DB   '+'          *Add one
0254 8801               DW   CMDPS
0256 2D                 DB   '-'          *Subtract one
0257 C801               DW   CMDMS
0259 41                 DB   'A'          *Accumulator
025A D601               DW   CMDA
                000A    TBLSZ EQU  (*-TABL)/3
                *
                *   *** INIT ***
                *
                * Initilize the robot.
                *
025C C9         INIT    RET             *Baptize
                025D    LAST EQU  *     *the robot
                        END
```

OBJECT CODE OF "THE MIND"

```
0000 :  3100 20CD  5C02 AFD3  C821 00CC  3ED0 3620
0010 :  23BC C20E  0021 413D  223A CC21  3030 223C
0020 :  CC22 3ECC  31FE 1F21  00CC 063A  3620 2305
0030 :  C22C 000E  7FCD AE00  CA2A 0060  6811 00CC
0040 :  CD50 0021  00CC 5058  06FF CD50  00C3 2700
0050 :  CD81 00DB  00E6 80C2  2400 E5CD  C600 78C2
0060 :  6500 B7E3  C9B7 E1C8  E5CD AE00  C27F 00E5
0070 :  2163 FD39  7CB7 FA24  00E1 2323  CD50 00E1
0080 :  C97B E63F  FE39 D224  00B2 CA90  003E 7F12
0090 :  7CB5 C2A5  00CD 4601  DB00 E680  CA95 00DB
00A0 :  014F C3A7  004E 237A  B3C8 7912  13C9 D5C5
00B0 :  2140 CC11  4000 060F  79BE CAC3  0019 05C2
00C0 :  B900 04C1  D1C9 C506  0A21 3E02  79BE 23CA
00D0 :  E600 2323  05C2 CD00  C121 1702  3E39 B9D8
00E0 :  79D6 30D8  BFC9 7E23  666F C1C9  C9CD 8100
00F0 :  E5CD C600  CA24 0078  B7C2 0001  E1C3 5000
0100 :  CDAE 00C2  0801 367F  C50E 20CD  AE00 C224
0110 :  00C1 7123  363D 23EB  E306 00CD  8100 3E20
0120 :  B9C2 2901  1B1B 121B  12CD 5300  D106 FFC9
0130 :  CD81 0079  FE29 C8CD  5300 C330  01CA 5C01
0140 :  CD46 01C3  7001 E52A  FE1F 297D  D252 01EE
0150 :  2D6F 22FE  1FE1 E601  C9C2 6201  CD50 00C3
0160 :  5000 E52A  3CCC 7CB5  2A3E CCB4  B5E1 FE30
0170 :  CA7E 01CD  5000 0600  CD50 0006  FFC9 0600
0180 :  CD50 0006  FFC3 5000  C8E5 CDE5  01E1 3E27
0190 :  BCC2 9C01  3E0F BDC2  9C01 E1C9  230E 0829
01A0 :  7CD6 64DA  A801 6723  0DC2 9F01  4C7D 213C
01B0 :  CCCD BB01  2379 CDBB  01E1 C936  2F34 D60A
01C0 :  D2BD 01C6  3A23 77C9  C8E5 CDE5  01E1 7CB5
01D0 :  2BC2 9D01  E1C9 CA50  00E5 CDE5  01E1 E3CD
01E0 :  8100 C323  0221 3CCC  CD81 00E5  2100 00D5
01F0 :  3E18 BCD2  F701 6754  5D29 2919  2916 0079
0200 :  D630 5F19  D1E3 CD81  00CD DC00  E3CA EF01
0210 :  3333 E33B  3BE3 C9CD  EB01 78B7  C223 02F1
0220 :  C353 00E3  7CB5 2BC2  3302 E106  00CD 5300
0230 :  06FF C9E3  E5C5 CD53  00C1 E1C3  2302 20EC
0240 :  0044 ED00  2830 0129  2400 7F24  003F 3D01
0250 :  5459 012B  8801 2DC8  0141 D601  C9
```